

Proving Termination Starting from the End

Pierre Ganty¹ and Samir Genaim²

¹ IMDEA Software Institute, Madrid, Spain

² Universidad Complutense de Madrid, Spain

Abstract. We present a novel technique for proving program termination which introduces a new dimension of modularity. Existing techniques use the program to incrementally construct a termination proof. While the proof keeps changing, the program remains the same. Our technique goes a step further. We show how to use the current partial proof to partition the transition relation into those behaviors known to be terminating from the current proof, and those whose status (terminating or not) is not known yet. This partition enables a new and unexplored dimension of incremental reasoning on the program side. In addition, we show that our approach naturally applies to conditional termination which searches for a precondition ensuring termination. We further report on a prototype implementation that advances the state-of-the-art on the grounds of termination and conditional termination.

1 Introduction

The question of whether or not a given program has an infinite execution is a fundamental theoretical question in computer science but also a highly interesting question for software practitioners. The first major result is that of Alan Turing, showing that the *termination problem* is undecidable. Mathematically, the termination problem for a given program *Prog* is equivalent to deciding whether the transition relation R induced by *Prog* is well-founded.

The starting point of our paper, is a result showing that the well-foundedness problem of a given relation R is equivalent to the problem of asking whether the transitive closure of R , noted R^+ , is disjunctively well-founded [24]. That is whether R^+ is included in some W (in which case W is called a *transition invariant*) such that $W = W_1 \cup \dots \cup W_n$, $n \in \mathbb{N}$ and each W_i is well-founded (in which case W is said to be *disjunctively well-founded*). This result has important practical consequences because it triggered the emergence of effective techniques, based on transition invariants, to solve the termination problem for real-world programs [11,2,28,20].

By replacing the well-foundedness problem of R with the equivalent disjunctive well-foundedness problem of R^+ , one allows for the incremental construction of W : when the inclusion of R^+ into W fails then use the information from the failure to update W with a further well-founded relation [10]. Although the proof is incremental for W , it is important to note that a similar result does not hold for R . That is, it is in general not true that given $R = R_1 \cup R_2$, if $R_1^+ \subseteq W$ and $R_2^+ \subseteq W$ then $R^+ \subseteq W$.

We introduce a new technique that, besides being incremental for W , further partitions the transition relation R separating those behaviors known to be terminating from

the current W , from those whose status (terminating or not) is not known yet. Formally, given R and a candidate W , we shall see how to compute a partition $\{R_G, R_B\}$ of R such that (a) $R_G^+ \subseteq W$; and (b) every infinite sequence $s_1 R s_2 R \cdots s_i R s_{i+1} \cdots$ (or *trace*) has a suffix that exclusively consists of transitions from R_B , namely we have $s_z R_B s_{z+1} R_B \cdots$ for some $z \geq 1$.

It follows that well-foundedness of R_B implies that of R . Consequently, we can focus our effort exclusively on proving well-foundedness of R_B . In the affirmative, then so is R and hence termination is proven. In the negative, then we have found an infinite trace in R_B , hence in R . We observed that working with R_B typically provides further hints on which well-founded relations to add to W . The partition of R into $\{R_G, R_B\}$ enables a new and unexplored dimension of modularity for termination proofs.

Let us mention that the partitioning of R is the result of adopting a fixpoint centric view on the disjunctive well-foundedness problem and leverage equivalent formulation of the inclusion check. More precisely, we introduce the dual of the check $R^+ \subseteq W$ by defining the adjoint to the function $\lambda X. X \circ R$ used to define R^+ . Without defining it now, we write the dual check as follows: $R \subseteq W^-$. We shall see that while the failure of $R^+ \subseteq W$ provides information to update W ; the failure of $R \subseteq W^-$ provides information on *all pairs* in R responsible for the failure of W as a transition invariant. This is exactly that information, of semantical rather than syntactical nature, that we use to partition R .

We show that the partitioning of R can be used not only for termination, but it also serves for conditional termination. The goal here is to compute a precondition, that is a set \mathcal{P} of states, such that no infinite trace starts from a state of \mathcal{P} . We show how to compute a (non-trivial) precondition from the relation R_B .

Our contributions are summarized as follows: (i) we present *Acabar*, a new algorithm which allows for enhanced modular reasoning about infinite behaviors of programs; (ii) we show that, besides termination, *Acabar* can be used in the context of conditional termination; and (iii) finally, we report on a prototype implementation of our techniques and compare it with the state-of-the-art on two grounds: the termination problem, and the problem of inferring a precondition that guarantees termination.

2 Example

In this section, we informally overview our proposed techniques on an example taken from the literature [9]. Consider the following loop:

```
while ( x > 0 ) { x := x + y; y := y + z; }
```

represented by the transition relation $R = \{x > 0, x' = x + y, y' = y + z, z' = z\}$, where the primed variables represent the values of the program variables after executing the loop body. Note that, depending on the input values, the program may not terminate (e.g. for $x = 1, y = 1$ and $z = 1$). Below we apply *Acabar* to prove termination. As we will see, this attempt ends with a failure which provide information on which subset of the transition relation to blame. Then, we will explain how to compute a termination precondition from this subset.

In order to prove termination of this loop, we seek a disjunctive well-founded relation W such that $R^+ \subseteq W$. To find such a W , *Acabar* is supported by incrementally (and automatically) inferring (potential) linear ranking functions for R or

R^+ [9,10]. When running on R , Acabar first adds the candidate well-founded relation $W_1 = \{x' < x, x > 0\}$ to W which is initially empty. Relation W_1 stems from the observation that, in R , x is bounded from below (as shown by the guard) but not necessarily decreasing. Hence, using $W = W_1$, Acabar partitions R into $\{R_G^{(1)}, R_B^{(1)}\}$ where:

$$\begin{aligned} R_G^{(1)} &= \{x > 0, x' = x + y, y' = y + z, z' = z, y < 0, z \leq 0\} \\ R_B^{(1)} &= \{x > 0, x' = x + y, y' = y + z, z' = z, y < 0, z > 0\} \vee \\ &\quad \{x > 0, x' = x + y, y' = y + z, z' = z, y \geq 0\} . \end{aligned}$$

The partition comes with the further guarantee that *every infinite trace in R must have a suffix that exclusively consists of transitions from $R_B^{(1)}$* , which means that if $R_B^{(1)}$ is well-founded then so is R . In addition, one can easily see that $(R_G^{(1)})^+ \subseteq W$.

Next, Acabar calls itself recursively on $R_B^{(1)}$ to show its well-foundedness. As before, it first adds $W_2 = \{y' < y, y \geq 0\}$ to W . Similarly to the construction of W_1 , W_2 stems from the observation that, in some parts of $R_B^{(1)}$, y is bounded from below but not necessarily decreasing. Then, using $W = W_1 \vee W_2$, Acabar partitions $R_B^{(1)}$ into:

$$\begin{aligned} R_G^{(2)} &= \{x > 0, x' = x + y, y' = y + z, z' = z, z < 0\} \\ R_B^{(2)} &= \{x > 0, x' = x + y, y' = y + z, z' = z, y \geq 0, z \geq 0\} . \end{aligned}$$

Again the partition $\{R_G^{(2)}, R_B^{(2)}\}$ of $R_B^{(1)}$ comes with a similar guarantee. This time it holds that every infinite trace in R must have a suffix that exclusively consists of transitions from $R_B^{(2)}$. Recursively applying Acabar on $R_B^{(2)}$ does not yield any further partitioning, that is $R_B^{(3)} = R_B^{(2)}$. The reason being that no potential ranking function is automatically inferred. Thus, Acabar fails to prove well-foundedness of R , which is indeed not well-founded. However, due to the above guarantee, we can use $R_B^{(2)}$ to infer a sufficient precondition for the termination of R . We explain this next.

Inferring a sufficient precondition is done in two steps: (i) we infer (an overapproximation of) the set of all states \mathcal{Z} visited by some infinite sequence of steps in $R_B^{(2)}$; and (ii) we infer (an overapproximation of) the set of all states \mathcal{V} each of which can reach \mathcal{Z} through some steps in R . Turning to the example, we infer $\mathcal{Z} = \{x > 0, y \geq 0, z \geq 0\}$ and the following overapproximation \mathcal{V}' of \mathcal{V} :

$$\mathcal{V}' = \{x \geq 1, z = 0, y \geq 0\} \vee \{x \geq 1, z \geq 1, x + y \geq 1, x + 2y + z \geq 1, x + 3y + 3z \geq 1\} .$$

It can be seen that every infinite trace visits only states in \mathcal{V}' , hence the complement of \mathcal{V}' is a precondition for termination.

Let us conclude this section by commenting on an example for which Acabar proves termination. Assume that we append $z := z - 1$ to the loop body above and call R' the induced transition relation. Following our previous explanations, running Acabar on R' updates W from \emptyset to W_1 , and then to $W_1 \vee W_2$. Then, and contrary to the previous explanations, Acabar will further update W to $W_1 \vee W_2 \vee W_3$ where W_3 is the well-founded relation $\{z' < z, z \geq 0\}$. From there, Acabar returns with value $R_B^{(3)} = \emptyset$, hence we have that R' is well-founded.

3 Preliminaries

A *transition system* is a pair (Q, R) where Q is the set of *states* and $R \subseteq Q \times Q$ is the *transition relation*. An *initialized* transition system includes a further component $I \subseteq Q$, the set of *initial states*. For simplicity, we defer the treatment of initial states to Sec. 8.

An *R-trace* is a sequence s_1, s_2, \dots, s_n of states such that for every i , $1 \leq i < n$ we have $(s_i, s_{i+1}) \in R$. When R is clear from the context we simply say *trace*. An *infinite R-trace* is a sequence s_1, s_2, \dots of states such that for every $i \geq 1$ we have $(s_i, s_{i+1}) \in R$. Given $R' \subseteq R$ and an infinite R -trace π we say that π has *infinitely many steps* in R' if $(s_i, s_{i+1}) \in R'$ for infinitely many $i \geq 1$.

Given a relation $R' \subseteq R$ and a set $Q' \subseteq Q$, define $\text{post}[R'](Q') \stackrel{\text{def}}{=} \{s' \in Q \mid \exists s \in Q' : (s, s') \in R'\}$. We say that this operator computes the *R'-successors* of Q' . Dually, define $\text{pre}[R'](Q') \stackrel{\text{def}}{=} \text{post}[R'^{-1}](Q') = \{s \in Q \mid \exists s' \in Q' : (s, s') \in R'\}$. We say that this operator computes the *R'-predecessors* of Q' .³

A relation $W \subseteq Q \times Q$ is called *disjunctively well-founded* iff W coincides with the union of finitely many relations (viz. $W = W_1 \cup \dots \cup W_n$) each of which is well-founded (viz. there is no infinite sequence s_1, s_2, \dots such that $(s_i, s_{i+1}) \in W_\ell$ for all $i \geq 1$).

In this paper, we adhere to the following conventions: calligraphic letters $\mathcal{X}, \mathcal{Y}, \dots$ refer to subsets of Q and capital letters X, Y, \dots refer to relations over Q , that is subsets of $Q \times Q$. Further, throughout the paper the letter W is used to denote a relation over Q that is disjunctively well-founded.

A *linear expression* is of the form $a_0 + a_1x_1 + \dots + a_nx_n$ where $a_i \in \mathbb{Z}$ and $\bar{x} = \langle x_1, \dots, x_n \rangle$ are *variables* ranging over \mathbb{Z} . An *atomic linear constraint* c is of the form $e_1 \text{ op } e_2$ where e_i is a linear expression and $\text{op} \in \{=, \geq, \leq, >, <\}$. A *formula* ψ is a Boolean combination of atomic linear constraints. Note that $\neg\psi$ is also a formula. For the sake of simplicity, a conjunction $c_1 \wedge \dots \wedge c_n$ of atomic linear constraints is sometimes written as the set $\{c_1, \dots, c_n\}$. A *solution* of a formula ψ is a mapping from its variables into the integers such that the formula evaluates to true. Sets and relations over, respectively, \mathbb{Z}^n and $\mathbb{Z}^n \times \mathbb{Z}^n$ are sometimes specified using formulas, with the customary convention, for relations, of variables and primed variables. For instance, the formula $\{x \geq 0, x' = x - y, y' = y\}$ defines the relation $R \subseteq \mathbb{Z}^2 \times \mathbb{Z}^2$ such that $R = \{\langle (x, y), (x', y') \rangle \mid x \geq 0 \wedge x' = x - y \wedge y' = y\}$.

Finally, we briefly recall classical results of lattice theory and refer to the classical book of Davey and Priestley [15] for further information. Let f be a function over a partially ordered set (L, \sqsubseteq) . A *fixpoint* of f is an element $l \in L$ such that $f(l) = l$. We denote by $\text{lfp } f$ and $\text{gfp } f$, respectively, the *least* and the *greatest fixpoint*, when they exist, of f . The well-known Knaster-Tarski's theorem states that each order-preserving function $f \in L \rightarrow L$ over a complete lattice $\langle L, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ admits a least (greatest) fixpoint and the following characterization holds:

$$\text{lfp } f = \sqcap \{x \in L \mid f(x) \sqsubseteq x\} \qquad \text{gfp } f = \sqcup \{x \in L \mid x \sqsubseteq f(x)\} . \quad (1)$$

³ We define R^{-1} , R^* and R^+ to be $R^{-1} = \{(s', s) \mid (s, s') \in R\}$, $R^* = \bigcup_{i \geq 0} R^i$ and $R^+ = R \circ R^*$ where R^0 is the identity, $R^{i+1} = R^i \circ R$ and $R_1 \circ R_2 = \{(s, s'') \mid \exists s' : (s, s') \in R_1 \wedge (s', s'') \in R_2\}$.

4 Modular Reasoning For Termination

A termination proof based on transition invariants consists in establishing the existence of a disjunctively well-founded transition invariant. That is, the goal is to prove the inclusion of R^+ , into some W .⁴ For short, we write $R^+ \subseteq W$. Proving termination is thus reduced to finding some W and prove that the inclusion hold.

In the above inclusion check, R^+ coincides with the least fixpoint of the function $\lambda Y. R \cup g(Y)$ where $g \stackrel{\text{def}}{=} \lambda Y. Y \circ R$. It is known [13] that if we can find an *adjoint function* \tilde{g} to g such that $g(X) \subseteq Y$ iff $X \subseteq \tilde{g}(Y)$ for all X, Y then there exists an equivalent inclusion check to $R^+ \subseteq W$. This equivalent check, denoted $R \subseteq W^-$ in the introduction, is such that W^- is defined as a greatest fixpoint of the function $\lambda Y. W \cap \tilde{g}(Y)$. Next, we define $\tilde{g} \stackrel{\text{def}}{=} \lambda Y. \neg(\neg Y \circ R^{-1})$.

Lemma 1. *Let X, Y be subsets of $Q \times Q$ we have: $X \circ R \subseteq Y \Leftrightarrow X \subseteq \neg(\neg Y \circ R^{-1})$.*

Proof. First we need an easily proved logical equivalence:

$$(\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3 \text{ iff } (\neg\varphi_3 \wedge \varphi_2) \Rightarrow \neg\varphi_1 .$$

Then we have:

$$\begin{aligned} & X \circ R \subseteq Y \\ \text{iff } \forall s, s', s_1 : ((s, s_1) \in X \wedge (s_1, s') \in R) & \Rightarrow (s, s') \in Y \\ \text{iff } \forall s, s', s_1 : ((s, s') \notin Y \wedge (s_1, s') \in R) & \Rightarrow (s, s_1) \notin X & \text{by above equivalence} \\ \text{iff } \forall s, s', s_1 : ((s, s') \notin Y \wedge (s', s_1) \in R^{-1}) & \Rightarrow (s, s_1) \notin X & \text{def. of } R^{-1} \\ \text{iff } \forall s, s', s_1 : ((s, s') \in \neg Y \wedge (s', s_1) \in R^{-1}) & \Rightarrow (s, s_1) \in \neg X \\ \text{iff } (\neg Y \circ R^{-1}) \subseteq \neg X \\ \text{iff } X \subseteq \neg(\neg Y \circ R^{-1}) & \quad \square \end{aligned}$$

Intuitively, g corresponds to *forward reasoning* for proving termination while \tilde{g} corresponds to *backward reasoning* because of the composition with R^{-1} . The least fixpoint $\text{lfp } \lambda Y. R \cup g(Y)$ is the least relation Z containing R and closed by composition with R , viz. $R \subseteq Z$ and $Z \circ R \subseteq Z$. On the other hand, the greatest fixpoint $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ is best understood as the result of removing from W all those pairs (s, s') of states such that $(s, s') \circ R^+ \not\subseteq W$. This process returns the largest subset Z' of W which is closed by composition with R , viz. $Z' \subseteq W$ and $Z' \circ R \subseteq Z'$. Using the results of Cousot [13] we find next that termination can be shown by proving either inclusion of Lem. 2.

Lemma 2 (from [13]). *$\text{lfp } \lambda Y. R \cup g(Y) \subseteq W \Leftrightarrow R \subseteq \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$.*

Proof.

$$\begin{aligned} \text{lfp } \lambda Y. R \cup g(Y) \subseteq W & \text{ iff } \exists A : R \subseteq A \wedge g(A) \subseteq A \wedge A \subseteq W & \text{by (1)} \\ & \text{ iff } \exists A : R \subseteq A \wedge A \subseteq \tilde{g}(A) \wedge A \subseteq W & \text{Lem. 1} \\ & \text{ iff } R \subseteq \text{gfp } \lambda Y. W \cap \tilde{g}(Y) & \text{by (1) } \square \end{aligned}$$

⁴ Recall that W is always assumed to be disjunctively well-founded.

As we shall see, the inclusion check based on the greatest fixpoint has interesting consequences when trying to prove termination.

An important feature when proving termination using transition invariants is to define actions to take when the inclusion check $\text{lfp } \lambda Y. R \cup g(Y) \subseteq W$ fails. In this case, some information is extracted from the failure (e.g., a counter example), and is used to enrich W with more well-founded relations [10].

We shall see that, for the backward approach, failure of $R \subseteq \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ induces a partition of the transition relation R into $\{R_G, R_B\}$ such that (a) $(R_G)^+ \subseteq W$; together with the following termination guarantee (b) every infinite R -trace contains a suffix that is an infinite R_B -trace (Lem. 4). An important consequence of this is that we can focus our effort exclusively on proving termination of R_B . It is important to note that the guarantee that no infinite R -trace contains infinitely many steps from R_G is not true for any partition $\{R_G, R_B\}$ of R but it is true for our partition which we define next.

Definition 1. Let $G = \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$, we define $\{R_G, R_B\}$ to be the partition of R given by $R_G = R \cap G$ and $R_B = R \setminus R_G$.

Example 1. Let $R = \{x \geq 1, x' = x + y, y' = y - 1\}$ and assume $W = \{x' < x, x \geq 1\}$ which is well-founded, hence disjunctively well-founded as well. Evaluating the greatest fixpoint (we omit calculations) yields

$$\begin{aligned} R_G &= \{x \geq 1, x' = x + y, y' = y - 1, y < 0\} \\ R_B &= \{x \geq 1, x' = x + y, y' = y - 1, y \geq 0\} \end{aligned}$$

which is clearly a partition of R . The relation R_G consists of those pairs of states where y is negative, hence x is decreasing as captured by W . On the other hand, R_B consists of those pairs where y is positive or null. It follows that, when taking a step from R_B , x does not decrease. This is precisely for those pairs that W fails to show termination. ■

Next, we state and prove the termination guarantees of the partition $\{R_G, R_B\}$.

Lemma 3. Given R_G as in Def. 1 we have $\text{lfp } \lambda Y. R_G \cup Y \circ R \subseteq W$.

Proof.

$$\begin{array}{ll} G \subseteq \tilde{g}(G) \wedge G \subseteq W & \text{def. of } G \text{ and (1)} \\ \text{only if } g(G) \subseteq G \wedge G \subseteq W & \text{Lem. 1} \\ \text{only if } R \cap G \subseteq G \wedge g(G) \subseteq G \wedge G \subseteq W & \\ \text{only if } R_G \subseteq G \wedge g(G) \subseteq G \wedge G \subseteq W & \text{def. of } R_G \\ \text{only if } \text{lfp } \lambda Y. R_G \cup g(Y) \subseteq W & \text{by (1) } \square \end{array}$$

An equivalent formulation of the previous result is $R_G \circ R^* \subseteq W$, which in turn implies, since $R_G \subseteq R$, that $(R_G \circ R^*)^+ \subseteq W$, and also $(R_G)^+ \subseteq W$.

Lemma 4. Every infinite R -trace has a suffix that is an infinite R_B -trace.

Proof. Assume the contrary, i.e., there exists an infinite R -trace s_1, s_2, \dots that contains infinitely many steps from R_G . Let $S = s_{i_1}, s_{i_2}, \dots$ be the infinite subsequence of states such that $(s_{i_j}, s_{i_{j+1}}) \in R_G$ for all $j \geq 1$. Recall also that $W = W_1 \cup \dots \cup W_n$ where each W_ℓ is well-founded. For any $s_i, s_j \in S$ with $i < j$ it holds that $(s_i, s_j) \in R_G \circ R^*$, and thus, according to Lem. 3, we also have that $(s_i, s_j) \in W_\ell$ for some $1 \leq \ell \leq n$. Ramsey's theorem [25] guarantees the existence of an infinite subsequence $S' = s_{j_1}, s_{j_2}, \dots$ of S , and a single W_ℓ , such that for all $s_i, s_j \in S'$ with $i < j$ we have $(s_i, s_j) \in W_\ell$. This contradicts that W_ℓ is well-founded and we are done. \square

Remark 1. When fixpoints are not computable, they can be approximated from above or from below [14]. It is routine to check that the results of Lemmas 3 and 4 remain valid when replacing $G = \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ in Def. 1 with $G' \subseteq \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$. Therefore we have that, even when approximating $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ from below, the termination guarantees of $\{R_G, R_B\}$ still hold. In Sec. 6, we shall see how to exploit this result in practice.

Example 2 (cont'd from Ex. 1). We left Ex. 1 with $W = \{x' < x, x \geq 1\}$ and $R_B = \{x \geq 1, x' = x + y, y' = y - 1, y \geq 0\}$. As argued previously, to prove the well-foundedness of R it is enough to show that R_B is well-founded. For clarity, we rename R_B into $R_B^{(1)}$. Next we partition $R_B^{(1)}$ as we did it for R in Ex. 1. As a result, we update W by adding the well-founded relation $\{y' < y, y \geq 0\}$. Then we evaluate again G (we omit calculations) which yields $R_B^{(2)} = \emptyset$. Hence we conclude from Lem. 4 that R is well-founded. \blacksquare

Building upon all the previous results, we introduce Acabar that is given at Alg. 1. Acabar is a recursive procedure that takes as input two parameters: a transition relation R and a disjunctively well-founded relation W . The second parameter is intended for recursive calls, hence the user should invoke Acabar as follows: $\text{Acabar}(R, \emptyset)$. We call it the *root call*. Upon termination, Acabar returns a subset R_B of the transition relation R . If it returns the empty set, then the relation R is well-founded, hence termination is proven. Otherwise ($R_B \neq \emptyset$), we can not know for sure if R is well-founded: there might be an infinite R -trace. However, Lem. 4 tells us that every infinite R -trace must have a suffix that is an infinite R_B -trace. It may also be the case that R_B is well-founded (and so is R) in which case it was not discovered by Acabar. Another case is that $R = R_B$. In this case we have made no progress and therefore we stop. Whenever $R_B \neq \emptyset$, we call this returned value the *problematic* subset of R .

Next we study progress properties of Acabar. We start by defining the sequence $\{R^{(i)}\}_{i \geq 0}$ where each $R^{(i)}$ is the argument passed to the i -th recursive call to Acabar. In particular, $R^{(0)}$ is the argument of the root call. Furthermore, we define the sequences $\{R_B^{(i)}\}_{i \geq 1}$ and $\{R_G^{(i)}\}_{i \geq 1}$ where $\{R_G^{(i)}, R_B^{(i)}\}$ is a partition of $R^{(i-1)}$ and $R_B^{(i)} = R^{(i)}$ for all $i \geq 1$.

Lemma 5. *Let a run of Acabar with at least $i \geq 1$ recursive calls, then we have*

$$R^{(0)} \supsetneq R^{(1)} \supsetneq \dots \supsetneq R^{(i)}.$$

Proof. The proof is by induction on i , for $i = 1$ it follows from the definitions that $R^{(1)} = R_B^{(1)}$ and $\{R_B^{(1)}, R_G^{(1)}\}$ is a partition of $R^{(0)}$. Moreover, since at least $i = 1$ recursive calls take place we find that the condition of line 5 fails, meaning neither $R_B^{(1)}$ nor $R_G^{(1)}$ is empty, hence $R^{(1)}$ is a strict subset of $R^{(0)}$. The inductive case is similar. \square

Algorithm 1: Enhanced modular reasoning

Acabar(R, W)
Input: a relation $R \subseteq Q \times Q$
Input: a relation $W \subseteq Q \times Q$ such that W is disjunctively well-founded
Output: $R_B \subseteq R$

```

1 begin
2    $W := W \cup \text{find\_dwf\_candidate}(R)$ 
3   let  $G$  be such that  $G \subseteq \text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ 
4    $R_B := R \setminus G$ 
5   if  $R_B = \emptyset$  or  $R_B = R$  then
6     return  $R_B$ 
7   else
8     return Acabar( $R_B, W$ )
  
```

By Lemmas 4 and 5, we have that every infinite $R^{(0)}$ -trace has a suffix that is an infinite $R_B^{(i)}$ -trace for every $i \geq 1$. As a consequence, forcing **Acabar** to execute line 6 after predefined number of recursive calls, it returns a relation $R_B^{(i)}$ such that the previous property holds. Incidentally, we find that **Acabar** proves program termination when it returns the empty set as stated next.

Theorem 1. *Upon termination of the call **Acabar**(R, \emptyset), if it returns the empty set, then the relation R is well-founded.*

Let us now turn to line 2. There, **Acabar** calls a subroutine **find_dwf_candidate**(R) implementing a heuristic search which returns a disjunctively well-founded relation using hints from the representation and the domain of R . Details about its implementation, that is inspired from previous work [9,10], will be given at Sec. 7 — we will consider the case of R being a relation over the integers of the form $R = \rho_1 \vee \dots \vee \rho_n$ where each ρ_i is a conjunction of linear constraints over the variables \bar{x} and \bar{x}' . Let us intuitively explain this procedure on an example.

Example 3 (cont'd from Ex. 2). **Acabar**(R, \emptyset) updates W as follows: (1) \emptyset ; (2) $\{x' < x, x \geq 1\}$; (3) $\{x' < x, x \geq 1\}, \{y' < y, y \geq 0\}$. The first update from \emptyset to $\{x' < x, x \geq 1\}$ is the result of calling **find_dwf_candidate**(R). The hint used by **find_dwf_candidate** is that x is bounded from below in R . The second update to W results from calling **find_dwf_candidate**($R_B = \{x \geq 1, x' = x + y, y' = y - 1, y \geq 0\}$). Since R_B has the linear ranking function $f(x, y) = y$, **find_dwf_candidate** returns $\{y' < y, y \geq 0\}$. ■

5 Acabar for Conditional Termination

As mentioned previously, upon termination, **Acabar** returns a subset R_B of the transition relation R . If this set is empty then R is well-founded and we are done. Otherwise, R_B is a non-empty subset and called the problematic set. In this section, we shall see how to compute, given the problematic set, a *precondition* \mathcal{P} for termination. More precisely,

\mathcal{P} is a set of states such that no infinite R -trace starts with a state of \mathcal{P} . We illustrate our definitions using the simple but challenging example of Sec. 2.

Example 4. Consider again the relation $R = \{x > 0, x' = x + y, y' = y + z, z' = z\}$. Upon termination Acabar returns the following relation:

$$R_B = \{x' = x + y, y' = y + z, z' = z, x > 0, y \geq 0, z \geq 0\}$$

which corresponds to all the cases where x is stable or increasing over time. ■

Lemma 4 tells us that every infinite R -trace π is such that $\pi = \pi_f \pi_\infty$ where π_f is a finite R -trace and π_∞ is an infinite R_B -trace. Our computation of a precondition for termination is divided into the following parts: (i) compute those states \mathcal{Z} visited by infinite R_B -trace; (ii) compute the set \mathcal{V} of R^* -predecessors of \mathcal{Z} , that is the set of states visited by some R -trace ending in \mathcal{Z} ; and (iii) compute \mathcal{P} as the complement of \mathcal{V} . Formally, (i) is given by a greatest fixpoint expression $\text{gfp } \lambda X. \text{pre}[R_B](X)$. This expression is directly inspired by the work of Bozga et al. [5] on deciding conditional termination. This greatest fixpoint is the largest set \mathcal{Z} of states each of which has an R_B -successor in \mathcal{Z} . Because of this property, every infinite R_B -trace visits only states in \mathcal{Z} . In $\pi = \pi_f \pi_\infty$, this corresponds to the suffix π_∞ that is an infinite R_B -trace.

Example 5. For R_B as given in Ex. 4, we have that $\mathcal{Z} = \{z \geq 0, y \geq 0, x > 0\}$ which contains the following infinite R_B -trace:

$$(x = 1, y = 0, z = 0) R_B (x = 1, y = 0, z = 0) R_B (x = 1, y = 0, z = 0) R_B \dots \quad \blacksquare$$

Let us now turn to (ii), that is computing the set \mathcal{V} of R^* -predecessors of \mathcal{Z} . It is known that \mathcal{V} coincides with $\text{lfp } \lambda X. \mathcal{Z} \cup \text{pre}[R](X)$. Intuitively, we prepend to those infinite R_B -traces a finite R -trace. That is, prefixing π_f to π_∞ results in $\pi = \pi_f \pi_\infty$. Finally, step (iii) results into a precondition for termination \mathcal{P} obtained by complementing \mathcal{V} .

Example 6. Computing $\text{lfp } \lambda X. \mathcal{Z} \cup \text{pre}[R](X)$ for \mathcal{Z} as given in Ex. 5 and $R = \{x' = x + y, y' = y + z, z' = z, x > 0, y \geq 0, z \geq 0\}$ (Ex. 4) gives $\mathcal{V} = \mathcal{V}_1 \vee \mathcal{V}_2$ where

$$\begin{aligned} \mathcal{V}_1 &= \{x \geq 1, z = 0, y \geq 0\} \\ \mathcal{V}_2 &= \{x \geq 1, z \geq 1\} \cup \{x + i * y + j * z \geq 1 \mid i \geq 1, j = \sum_{k=0}^{i-1} k\} . \end{aligned}$$

Intuitively, the set \mathcal{V}_1 of states corresponds to entering the loop with $z = 0$ and y non-negative, in which case the loop clearly does not terminate. The set \mathcal{V}_2 of states corresponds to entering the loop with z positive, and the loop does not terminate after i -th iterations for all i . Note that \mathcal{V}_2 consists of infinitely many atomic formulas. Complementing \mathcal{V} gives \mathcal{P} . ■

Theorem 2. *There exists an infinite R -trace starting from s iff $s \notin \mathcal{P}$.*

Approximations. As argued previously, it is often the case that only approximations of fixpoints are available. In our case, any overapproximation of either \mathcal{Z} or \mathcal{V} can be exploited to infer \mathcal{P} . Because of approximations, we lose the if direction of the theorem, that is, we can only say that there is no infinite R -trace starting from some $s \in \mathcal{P}$.

Example 7. Using finite disjunctions of linear constraints, we can approximate \mathcal{V} by

$$\{x \geq 1, z = 0, y \geq 0\} \vee \{x \geq 1, z \geq 1, x + y \geq 1, x + 2y + z \geq 1, x + 3y + 3z \geq 1\}$$

and then the complement \mathcal{P} is

$$x \leq 0 \vee x + y \geq 1 \vee x + 2y + z \geq 1 \vee x + 3y + 3z \geq 1 \vee z \leq -1 \vee (y \leq -1 \wedge z \leq 0)$$

which is a sufficient precondition for termination. Note that the first 4 disjuncts correspond to the executions which terminates after 0, 1, 2 and 3 iterations. ■

6 Implementation

We have implemented the techniques described in Sec. 4 and 5 for the case of multiple-path integer linear-constraint loops. These loops correspond to relations of the form $R = \rho_1 \vee \dots \vee \rho_d$ where each ρ_i is a conjunction of linear constraints over the variables \bar{x} and \bar{x}' . In this context, the set Q of states is equal to \mathbb{Z}^n where n is the number of variables in \bar{x} . This is a classical setting for termination [4,6,24]. Internally, we represent sets of states and relations over them as DNF formulas where the atoms are linear constraints. In what follows, we explain sufficient implementation details so that our experiments can be independently reproduced if desired. Our implementation is available [1].

We start with line 2 of Alg. 1. Recall that the purpose of this line is to add more well-founded relations to W based on the current relation R . In our implementation, W consists of well-founded relations of the form $\{f(\bar{x}) \geq 0, f(\bar{x}') < f(\bar{x})\}$ where f is a linear function [10,9]. Thus, our implementation looks for such well-founded relations. In particular, for each ρ_i of R we add new well-founded relations to W as follows: if ρ_i has a linear ranking function $f(\bar{x})$ that is synthesized automatically [24,4] then $\{f(\bar{x}') < f(\bar{x}), f(\bar{x}) \geq 0\}$ is added to W ; otherwise, let $\{f_1(\bar{x}) \geq 0, \dots, f_d(\bar{x}) \geq 0\}$ be the result of projecting each ρ_i on \bar{x} (i.e., eliminating variables \bar{x}' from ρ_i), then $\{\{f_i(\bar{x}') < f_i(\bar{x}), f_i(\bar{x}) \geq 0\} \mid 1 \leq i \leq d\}$ is added to W . Because f_i is bounded but not necessarily decreasing, it is called a *potential linear ranking function* [9].

As for line 3, recall that G is a subset of $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$. Furthermore, the sole purpose of G is to compute $R_B = R \setminus G$. We now observe that $\neg G$, the complement of G , is as good as G . In fact, $R_B = R \cap (\neg G)$. So by considering $\neg G$ instead, what we are looking for is an overapproximation of $\neg(\text{gfp } \lambda Y. W \cap \tilde{g}(Y))$. Next we recall Park's theorem replacing the above expression by a least fixpoint expression.

Theorem 3 (From [23]). *Let $\langle L, \sqsubseteq, \sqcap, \sqcup, \top, \perp, \neg \rangle$ be a complete Boolean algebra and let $f \in L \rightarrow L$ be an order-preserving function then $f' = \lambda X. \neg(f(\neg X))$ is an order-preserving function on L and $\neg(\text{gfp } f) = \text{lfp } f'$.*

Park's theorem applies in our setting because computations are carried over the Boolean algebra $\langle 2^{(Q \times Q)}, \sqsubseteq, \cap, \cup, (Q \times Q), \emptyset, \neg \rangle$. Applying it to $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ where $\tilde{g}(Y) = \neg(\neg Y \circ R^{-1})$, we find that

$$\neg(\text{gfp } \lambda Y. W \cap \neg(\neg Y \circ R^{-1})) = \text{lfp } \lambda Y. (\neg W) \cup Y \circ R^{-1} .$$

Therefore, to implement line 3, we rely on abstract interpretation to compute an overapproximation of $\text{lfp } \lambda Y. (\neg W) \cup Y \circ R^{-1}$, hence, by negation, an underapproximation of $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ therefore complying with the requirement on G .

As far as abstract interpretation is concerned, our implementation uses a combination of predicate abstraction [18] and trace partitioning [22]. The set of predicates is given by a finite set of atomic linear constraints and is also closed under negation, e.g., if $x + y \geq 0$ is a predicate then $x + y \leq -1$ is also a predicate. Abstract values are positive Boolean combination of atoms taken from the set of predicates. Observe that although negation is forbidden in the definition of abstract values, the abstract domain is closed under complement.

The set of predicates is chosen so as the following invariant to hold: each time the control hits line 3, the set contains enough predicates to represent precisely each well-founded relation in W . Our implementation provides enhanced precision by enforcing a stronger invariant: besides the above predicates for W , it includes all atomic linear constraints occurring in the formulas representing X_1, \dots, X_ℓ where $\ell \geq 0$, $X_0 = (\neg W)$ and $X_{i+1} = (\neg W) \cup X_i \circ R^{-1}$. The value of ℓ is user-defined and, in our experiments, it did not exceed 1.

To further enhance precision at line 3, we apply trace partitioning [22]. The set of R -traces is partitioned using the linear atomic constraints of the form $f(\vec{x}') < f(\vec{x})$ that appear in W . More precisely, partitioning R on $f(\vec{x}') < f(\vec{x})$ is done by replacing each ρ_i by $(\rho_i \wedge f(\vec{x}') < f(\vec{x})) \vee (\rho_i \wedge f(\vec{x}') \geq f(\vec{x}))$.

As for conditional termination, overapproximating $\mathcal{Z} = \text{gfp } \lambda X. \text{pre}[R_B](X)$ is done by computing the last element X_ℓ from the finite sequence X_0, \dots, X_ℓ given by $X_0 = Q$ and $X_{i+1} = X_i \wedge \text{pre}[R_B](X_i)$ where ℓ is predefined. The result is always representable as DNF formula where the atoms can be any atomic linear constraints. As for $\mathcal{V} = \text{lfp } \lambda X. X_\ell \cup \text{pre}[R](X)$, an overapproximation is computed in a similar way to that of line 3, i.e., using a combination of predicate abstraction and trace partitioning.

7 Experiments

We have evaluated our prototype implementation against a set of benchmarks collected from publications in the area [9,7]. In what follows, we present the results of our implementation for those loops, and compare them to existing tools for proving termination [26,7,6] as well as tools for inferring preconditions for termination [9]. We compare the different techniques according to what the corresponding implementations report. We ignore performance because, for the selected benchmarks, little insight can be gained from performance measurements when an implementation was available (which was not always the case [27]).

The benchmarks accompanied with our results are depicted in Table 1. Translating each loop to a relation of the form $R = \rho_1 \vee \dots \vee \rho_n$ is straightforward. Every line in the table includes a loop and its inferred termination precondition (*true* means it terminates for any input). In addition, preconditions (different from *true*) marked with • are optimal, i.e., the corresponding loop is non-terminating for any state in the complement.

We have divided the benchmarks into 3 groups: (1–5), (6–15) and (16–41). With the exception of loop 1, each loop in group (1–5) includes non-terminating executions and

#	loop	termination precondition
1	while ($x \geq 0$) $x' = -2x + 10$;	<i>true</i>
2	while ($x > 0$) $x' = x + y$; $y' = y + z$;	$x \leq 0 \vee z < 0 \vee$ $(z = 0 \wedge y < 0) \vee$ $x + y \leq 0 \vee x + 2y + z \leq 0 \vee$ $x + 3y + 3z \leq 0$
3	while ($x \leq N$) if (*) { $x' = 2 * x + y$; $y' = y + 1$; } else $x' = x + 1$;	$x > n \vee x + y \geq 0$
4	@requires $n > 200$ and $y < 9$ while (1) if ($x < n$) { $x' = x + y$; if ($x' \geq 200$) break; }	$n \leq 200 \vee y \geq 9 \vee$ $(x < n \wedge y \geq 1) \vee$ $(x < n \wedge x \geq 200 \wedge x + y \geq 200)$
5	while ($x < y$) if ($x > y$) $x' = x - y$; else $y' = y - x$;	$\bullet (x \geq 1 \wedge y \geq 1) \vee x = y$
6	while ($x < 0$) $x' = x + y$; $y' = y - 1$;	$x \geq 0 \vee x + y \geq 0 \vee$ $x + 2y \geq 1 \vee x + 3y \geq 3$
7	while ($x > 0$) $x' = x + y$; $y' = -2y$;	$\bullet x \leq 0 \vee y \neq 0$
8	while ($x < y$) $x' = x + y$; $y' = -2y$;	$\bullet x \geq 0 \vee y \neq 0$
9	while ($x < y$) $x' = x + y$; $2y' = y$;	$\bullet x \geq 0 \vee y \neq 0$
10	while ($4x - 5y > 0$) $x' = 2x + 4y$; $y' = 4x$;	$\bullet 5y - 4x \geq 0 \vee$ $(3x - 4y \geq 0 \wedge 16x - 21y \geq 1)$
11	while ($x < 5$) $x' = x - y$; $y' = x + y$;	$\bullet x \neq 0 \vee y \neq 0$
12	while ($x > 0$ and $y > 0$) $x' = -2x + 10y$;	$\bullet x \leq 3 \vee 10y - 3x \neq 0$
13	while ($x > 0$) $x' = x + y$;	$x \leq 0 \vee y < 0 \vee x + y \leq 0$
14	while ($x < 10$) $x' = -y$; $y' = y + 1$;	$\bullet y \leq -10 \vee x \geq 10$
15	while ($x < 0$) $x' = x + z$; $y' = y + 1$; $z' = -2y$	$x \geq 0 \vee x + z \geq 0$
16	while ($x > 0$ and $x < 100$) $x' \geq 2x + 10$;	\star <i>true</i>
17	while ($x > 1$) $-2x' = x$;	\star <i>true</i>
18	while ($x > 1$) $2x' \leq x$;	\star <i>true</i>
19	while ($x > 0$) $2x' \leq x$;	\star <i>true</i>
20	while ($x > 0$) $x' = x + y$; $y' = y - 1$;	<i>true</i>
21	while ($4x + y > 0$) $x' = -2x + 4y$; $y' = 4x$;	$4x + y \leq 0 \vee$ $(x - 4x \geq 0 \wedge 8x - 15y \geq 1)$
22	while ($x > 0$ and $x < y$) $x' = 2x$; $y' = y + 1$;	<i>true</i>
23	while ($x > 0$) $x' = x - 2y$; $y' = y + 1$;	<i>true</i>
24	while ($x > 0$ and $x < n$) $x' = -x + y - 5$; $y' = 2y$; $n' = n$;	<i>true</i>
25	while ($x > 0$ and $y < 0$) $x' = x + y$; $y' = y - 1$;	\star <i>true</i>
26	while ($x - y > 0$) $x' = -x + y$; $y' = y + 1$;	<i>true</i>
27	while ($x > 0$) $x' = y$; $y' = y - 1$;	<i>true</i>
28	while ($x > 0$) $x' = x + y - 5$; $y' = -2y$;	<i>true</i>
29	while ($x + y > 0$) $x' = x - 1$; $y' = -2y$;	<i>true</i>
30	while ($x > y$) $x' = x - y$; $1 \leq y' \leq 2$	\star <i>true</i>
31	while ($x > 0$) $x' = x + y$; $y' = -y - 1$;	<i>true</i>
32	while ($x > 0$) $x' = y$; $y' \leq -y$;	\star <i>true</i>
33	while ($x < y$) $x' = x + 1$; $y' = z$; $z' = z$;	<i>true</i>
34	while ($x > 0$) $x' = x + y$; $y' = y + z$; $z' = z - 1$;	<i>true</i>
35	while ($x + y \geq 0$ and $x \leq z$) $x' = 2x + y$; $y' = y + 1$; $z' = z$	<i>true</i>
36	while ($x > 0$ and $x \leq z$) $x' = 2x + y$; $y' = y + 1$; $z' = z$	<i>true</i>
37	while ($x \geq 0$) $x' = x + y$; $y' = z$; $z' = -z - 1$;	<i>true</i>
38	while ($x - y > 0$) $x' = -x + y$; $y' = z$; $z' = z + 1$;	<i>true</i>
39	while ($x > 0$ and $x < y$) $x' > 2x$; $y' = z$; $z' = z$;	<i>true</i>
40	while ($x \geq 0$ and $x + y \geq 0$) $x' = x + y + z$; $y' = -z - 1$; $z' = z$;	\star <i>true</i>
41	while ($x + y \geq 0$ and $x \leq n$) $x' = 2x + y$; $y' = z$; $z' = z + 1$; $n' = n$;	<i>true</i>

Table 1. Benchmarks used in experiments. Loops (1–5) are taken from [9] and (6–41) from [7].

thus those loops are suitable for inferring preconditions. Our implementation reports the same preconditions as the tool of Cook et al. [9] save for loop 1 for which their tool is reported to infer the precondition $x > 5 \vee x < 0$, while we prove termination for all input. Note that every other tool used in the comparison [7,6,26] fail to prove termination of this loop. Further, the precondition we infer for loop 5 is optimal.

All the loops (6–15) are non-terminating. Chen et al. [7] report that their tool cannot handle them since it aims at proving termination and not inferring preconditions for termination. We infer preconditions for all of them, and in addition, most of them are optimal (those marked with \bullet). Unfortunately for those loops we could not compare with the tool of Cook et al. [9], since there is no implementation available [27].

Loops in the group (16–41) are all terminating. Those marked with \star actually have linear ranking functions, those unmarked require disjunctive well-founded transition invariants with more than one disjunct. We prove termination of all of them except loop 21. We point that the tool of Chen et al. [7] also fails to prove termination of loop 21, but also of loop 34. On the other benchmarks, they prove termination. They also report that PolyRank [6] failed to prove termination of any of the loops that do not have a linear ranking function. In addition, we applied ARMC [26] on the loops of the group (16–41). ARMC, a transition invariants based prover, succeeded to prove termination for all those loops with a linear ranking function (marked with \star) and also loop 39.

Next we discuss in details the analysis of two selected examples from Table 1.

Example 8. Let us explain the analysis of loop 1 in details starting with the root call $\text{Acabar}(R, \emptyset)$ where $R = \{x \geq 0, x' = -2x + 10\}$. At line 2, since R includes the bound $x \geq 0$, i.e., $f(x) = x$ is a potential linear ranking function, we add $\{x' < x, x \geq 0\}$ to W . Computing G at line 3, hence R_B at the following line, results in $R_B = \rho_1 \vee \rho_2$ where $\rho_1 = \{x' = -2x + 10, x \geq 0, x \leq 3\}$ and $\rho_2 = \{x' = -2x + 10, x \geq 4, x \leq 5\}$.

Note that ρ_1 is enabled for $0 \leq x \leq 3$ and in this case $x' > x$. Also ρ_2 is enabled for $x = 4$ or $x = 5$ for which $x' < x$ and thus $\rho_2 \subseteq W$, however, after one more iteration, the value of x increases (this is why ρ_2 is included in R_B). Transitions for which $x > 5$ are not included in R_B , hence they belong to R_G itself included in W (Lem. 3). Hence when $x > 5$ termination is guaranteed, this is also easily seen since those transitions terminate after one iteration.

Since R_B is neither empty nor equal to R , a recursive call to $\text{Acabar}(R_B, W)$ takes place. At line 2, we add $\{-x' < -x, 10 - x \geq 0\}$ to W since $f(x) = 10 - x$ is a linear ranking function for ρ_1 . Note that ρ_2 has the linear ranking function $f(x) = x$ already included in W . Computing G at line 3, hence R_B , yields $R_B = \emptyset$ and therefore we conclude that the loop terminates for any input. ■

Example 9. Let us explain the analysis of loop 9 in details starting with the root call $\text{Acabar}(R, \emptyset)$ where $R = \{x < y, x' = x + y, 2y' = y\}$. At line 2, since R includes the bound $y - x > 0$, i.e., $f(x, y) = y - x - 1$ is a potential linear ranking function, we add $\{y' - x' < y - x, y - x - 1 \geq 0\}$ to W . Computing G at line 3, hence R_B yields $R_B = \{x < y, x' = x + y, 2y' = y, y \leq 0\}$. Note that R_B exclusively consists of transitions where y is not positive, in which case $x' - y' \geq x - y$ and thus not included in W . Transitions where y is positive are not included in R_B (hence they belong to R_G) since they always decrease $x - y$, and thus are transitively included in W (Lem. 3).

Since R_B is neither empty nor equal to R , we call recursively $\text{Acabar}(R_B, W)$. At line 2, since R includes the bound $y \leq 0$ (or equivalently $-y \geq 0$), i.e., $f(x, y) = -y$ is a potential linear ranking function, we add $\{-y' < -y, -y \geq 0\}$ to W . Computing G at line 3, hence R_B yields $R_B = \{x < y, x' = x + y, 2y' = y, y = 0\}$. Note that R_B exclusively consists of transitions where $y = 0$, which keeps both values of x and y unchanged. Transitions in which y is negative belong to R_G , hence they are transitively covered by W (Lem. 3), in particular by the last update (viz. $\{-y' < -y, -y \geq 0\}$) to W .

Since R_B is neither empty nor equal to R , we call recursively $\text{Acabar}(R_B, W)$. This time our implementation does not further enrich W with a well-founded relation, and as a consequence, after computing G at line 3, we get that $R_B = R$. Hence, Acabar returns with $R_B = \{x < y, x' = x + y, 2y' = y, y = 0\}$.

Now, given R_B , we infer a precondition for termination as described in Sec. 5. We first compute $\text{gfp } \lambda X. \text{pre}[R_B](X)$, which in this case, converges in two steps with $\mathcal{Z} \equiv y = 0 \wedge x < 0$. Then we compute $\text{lfp } \lambda X. \mathcal{Z} \cup \text{pre}[R](X)$, which results in $\mathcal{V} \equiv y = 0 \wedge x < 0$. The complement, $\mathcal{P} \equiv y < 0 \vee y > 0 \vee x < 0$, is a precondition for termination. Note that the result is optimal, i.e., \mathcal{V} is a precondition for non-termination. Optimality is achieved because \mathcal{Z} and \mathcal{V} coincide with the gfp and the lfp of the corresponding operators, and are not overapproximations. ■

8 Conclusion

This work started with the invited talk of A. Podelski at ETAPS '11 who remarked that the inclusion check $R^+ \subseteq W$ is equivalently formulated as a safety verification problem where states are made of pairs. Back to late 2007, a PhD thesis [17] proposed a new approach to the safety verification problem in which the author shows how to leverage the equivalent backward and forward formulations of the inclusion check. Those two events planted the seeds for the backward inclusion check $R \subseteq W^-$, and later Acabar .

Initial States. For the sake of simplicity, we deliberately excluded the initial states \mathcal{I} from the previous developments. Next, we introduce two possible options to incorporate knowledge about the initial states in our framework. The first option consists in replacing R by R' that is given by $R \cap (\text{Acc} \times \text{Acc})$ where Acc denotes (an overapproximation of) the reachable states in the system. Formally, Acc is given by the least fixpoint $\text{lfp } \lambda X. \mathcal{I} \cup \text{post}[R](X)$.

The second option is inspired by the work of Cousot [12] where he mixes backward and forward reasoning. We give here some intuitions and preliminary development. Recall that the greatest fixpoint $\text{gfp } \lambda Y. W \cap \tilde{g}(Y)$ of line 3 is best understood as the result of removing all those pairs $(s, s') \in W$ such that $(s, s') \circ R^+ \not\subseteq W$. We observe that the knowledge about initial states is not used in the greatest fixpoint. A way to incorporate that knowledge is to replace the greatest fixpoint expression by the following one $\text{gfp } \lambda Y. (B \cap W) \cap \tilde{g}(Y)$ where B takes the reachable states into account. In a future work, we will formally develop those two options and evaluate their benefit.

Related Works. As for termination, our work is mostly related to the work of Cook et al. [10,11] where the inclusion check $R^+ \subseteq W$ [24] is put to work by incrementally constructing W . Our approach, being based on the dual check $R \subseteq W^-$, adds a new dimension of modularity/incrementality in which R is also modified to safely exclude

those transitions for which the current proof is sufficient. The advantage of the dual check was shown experimentally in Sec. 7. However, let us note that in our implementation we use potential ranking functions and trace partitioning, which are not used in ARMC [10]. Moreover, it smoothly applies to conditional termination.

Kroening et al. [20] introduced the notion of compositional transition invariants, and used it to develop techniques that avoid the performance bottleneck of previous approaches [11]. Recently, Chen et al. [7] proposed a technique for proving termination of single-path linear-constraint loops. Contrary to their techniques, we handle general transition relations and our approach applies also to conditional termination.

As for conditional termination, the work of Cook et al. [9] is the closest to ours. However, we differ in the following points: (a) we do not use universal quantifier elimination, whose complexity is usually very high, depending on the underlying theory used to specify R . Instead, we adapt a fixpoint centric view that allows using abstract interpretation, and thus to control precision and performance; (b) we do not need special treatment for loop with phase transitions (as the one of Sec. 2), they are handled transparently in our framework. Bozga et al. [5] studied the problem of deciding conditional termination. Their main interest is to identify family of systems for which $gfp\ \lambda X. pre[R](X)$, the set of non-terminating states, is computable.

It is worth “terminating” by mentioning that several formulations, of the termination problem, similar to the check $R^+ \subseteq W$ have appeared before [8,21,16]. They have also led to practical tools for corresponding programming paradigms. The relation between these approaches was recently studied [19]. Works based on these formulations, in particular those that construct global ranking functions for R [3], might serve as a starting point to understand some (completeness) properties of our approach. This is left for future work.

References

1. Acabar. <http://loopkiller.com/acabar>
2. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: COSTA: Design and implementation of a cost and termination analyzer for java bytecode. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects, FMCO’07. LNCS, vol. 5382, pp. 113–132. Springer (2007)
3. Ben-Amram, A.M.: Size-change termination, monotonicity constraints and ranking functions. In: CAV ’09: Proc. 21st Int. Conf. on Computer Aided Verification. pp. 109–123. LNCS, Springer (2009)
4. Ben-Amram, A.M., Genaim, S.: On the linear ranking problem for integer linear-constraint loops. In: POPL ’13: Proc. 40th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. ACM (2013), to appear
5. Bozga, M., Iosif, R., Konečný, F.: Deciding conditional termination. In: TACAS ’12: Proc. 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 7214, pp. 252–266. Springer (2012)
6. Bradley, A.R., Manna, Z., Sipma, H.B.: The polyranking principle. In: ICALP ’05: Proc. of 32nd Int. Colloquium on Automata, Languages and Programming. LNCS, vol. 3580, pp. 1349–1361. Springer (2005)
7. Chen, H.Y., Flur, S., Mukhopadhyay, S.: Termination proofs for linear simple loops. In: SAS ’09: Proc. 19th Int. Static Analysis Symp. LNCS, vol. 7460, pp. 422–438. Springer (2012)

8. Codish, M., Taboch, C.: A semantic basis for the termination analysis of logic programs. *J. Log. Program.* 41(1), 103–123 (1999)
9. Cook, B., Gulwani, S., Lev-Ami, T., Rybalchenko, A., Sagiv, M.: Proving conditional termination. In: *CAV '08: Proc. 20th Int. Conf. on Computer Aided Verification*. pp. 328–340. No. 5123 in LNCS, Springer (2008)
10. Cook, B., Podelski, A., Rybalchenko, A.: Abstraction refinement for termination. In: *SAS '05: Proc. 12th Int. Static Analysis Symp.* pp. 87–101. No. 3672 in LNCS, Springer (2005)
11. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: *PLDI '06: Proc. 27th ACM-SIGPLAN Conf. on Programming Language Design and Implementation*. pp. 415–426. ACM (2006)
12. Cousot, P.: Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble (March 1978)
13. Cousot, P.: Partial completeness of abstract fixpoint checking, invited paper. In: *SARA '00: Proc. 4th Int. Symp. on Abstraction, Reformulations and Approximation*. LNAI, vol. 1864, pp. 1–25. Springer (2000)
14. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *POPL '77: Proc. 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*. pp. 238–252. ACM Press (1977)
15. Davey, B.A., Priestley, H.A.: *Introduction to Lattices and Order*. Cambridge University Press, Cambridge (1989)
16. Dershowitz, N., Lindenstrauss, N., Sagiv, Y., Serebrenik, A.: A general framework for automatic termination analysis of logic programs. *Appl. Algebra Eng. Commun. Comput.* 12(1/2), 117–156 (2001)
17. Ganty, P.: *The Fixpoint Checking Problem: An Abstraction Refinement Perspective*. Ph.D. thesis, Université Libre de Bruxelles (2007)
18. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: *CAV '97: Proc. 9th Int. Conf. on Computer Aided Verification*. LNCS, vol. 1254, pp. 72–83. Springer (1997)
19. Heizmann, M., Jones, N.D., Podelski, A.: Size-change termination and transition invariants. In: *SAS '10: Proc. 20th Int. Static Analysis Symp.* pp. 22–50. LNCS, Springer (2010)
20. Kroening, D., Sharygina, N., Tsitovich, A., Wintersteiger, C.M.: Termination analysis with compositional transition invariants. In: *CAV '10: Proc. 20th Int. Conf. on Computer Aided Verification*. LNCS, vol. 6174, pp. 89–103. Springer (2010)
21. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: *POPL '01: Proc. 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*. pp. 81–92. ACM (2001)
22. Mauborgne, L., Rival, X.: Trace partitioning in abstract interpretation based static analyzers. In: *ESOP '05: Proc. 14th European Symp. on Programming*. LNCS, vol. 3444, pp. 5–20. Springer (2005)
23. Park, D.: Fixpoint induction and proofs of program properties. In: *Machine Intelligence*, vol. 5, pp. 59–78. American Elsevier (1969)
24. Podelski, A., Rybalchenko, A.: Transition invariants. In: *LICS '04: Proc. 19th Annual IEEE Symp. on Logic in Computer Science*. pp. 32–41. IEEE (2004)
25. Ramsey, F.P.: On a problem of formal logic. *London Math. Society* 30, 264–286 (1929)
26. Rybalchenko, A.: Armc. <http://www7.in.tum.de/~rybal/armc/> (2008)
27. Rybalchenko, A.: Personal communication (2012)
28. Spoto, F., Mesnard, F., Payet, É.: A termination analyzer for java bytecode based on path-length. *ACM Trans. Program. Lang. Syst.* 32(3) (2010)